Using Steady-State TCP Behavior for Proactive Queue Management

Sunil Thulasidasan Wu-chun Feng

Research and Development in Advanced Network Technology (RADIANT)
Advanced Computing Laboratory (CCS-1)
Los Alamos National Laboratory
Los Alamos, New Mexico

Abstract— *This paper describes and evaluates the use of steady-state TCP behavior knowledge for proactive queue management (PQM). We show how routers can use flow parameters and the knowledge of TCP end-host behavior to intelligently mark packets to prevent queue build-up, and thus prevent congestion from ever occurring, while at the same time provide high link utilization and low packet loss. This is in contrast to traditional active queue management (AQM) schemes, where incipient congestion is detected early and then reacted to. Using a queue management algorithm that tracks TCP behavior allows queue sizes to be deterministically stabilized at very low levels, irrespective of the number of flows passing through the router. An advantage of our algorithm is that there are no parameters that need to be tuned to achieve optimal performance in a given scenario, thus avoiding repeated trials with different parameters – an approach common to other AQM schemes. Further we show that our scheme out-performs the RED and BLUE AQM schemes with respect to queueing delay and performs as well with respect to link utilization and packet loss.*

Keywords— **Congestion control, active queue management, RED, BLUE.**

I. Introduction

ACTIVE **Q**ueue **M**anagement (AQM) for congestion avoidance is one of the most widely researched areas in networking. Congestion leads to lost packets, thus wasting all the resources that the packet may have consumed on its way from source to destination. A persistently large queue size at a router is regarded as an indicator of congestion. The basic idea behind AQM is to prevent queue build-up by detecting incipient congestion early on and notifying the TCP end host by dropping or marking packets. The typical behavior of TCP to a dropped packet is to reduce its congestion window (*cwnd*) in half [6]. *cwnd* is then gradually built up by additively increasing it by one packet each round-trip time (*RTT*) until the next packet drop occurs. This gives rise to the well-known "*sawtooth*" behavior of the TCP congestion window. Based on this behavior, researchers have been able to develop mathematical models on the steady-state behavior of TCP [7], [10]. In this paper, we show how applying one of these models at routers can help prevent queue build-up. Furthermore, queues can be deterministically stabilized at desired levels while simultaneously maintaining high link utilization. This quality is especially beneficial to real-time multimedia applications that are adversely affected by large queueing delays or delay jitter.

We evaluate the performance of our mechanism, called GREEN, against two well-known AQM schemes in routers, RED [4] and BLUE [1]. GREEN performs as well as RED and BLUE in terms of link utilization and packet loss, while greatly out-performing both in terms of queueing delay.

In what follows, we first describe related work on AQM in Section II. Section III introduces our algorithm and the mathematical model that serves as its basis. Section IV provides a detailed evaluation of the performance of GREEN in comparison to other AQM schemes. Section VI describes implementation details including methods for estimating flow parameters. Finally, Section VII concludes with the implications of this work and directions for future research.

II. Background

When a network is congested, connections compete for scarce link bandwidth. This leads to queue build-up at routers, and eventually to lost packets due to queue overflow. The TCP end host responds to this by reducing its congestion window in half. However, considerable time may pass before the sender detects a lost packet, while continuing to send at increasing rates prior to this detection. This leads to even more lost packets. AQM schemes try

This paper is Los Alamos Unclassified Report LA-UR 02-2777

to alleviate this problem by detecting incipient congestion early on and notifying the senders. In this paper we evaluate our algorithm against two well-known AQM implementations, RED and BLUE.

RED [4] is the most widely studied AQM scheme. RED uses an exponentially weighted moving average of the queue length, which is used as an indicator of congestion. When the average queue length exceeds a certain threshold, known as $Q_{minthresh}$, packets are randomly dropped or marked[1] with a probability that is proportional to the queue length. When the average queue size exceeds the maximum threshold, known as $Q_{maxthresh}$, all incoming packets are marked. Further, RED also prevents synchronization between sources[2] by avoiding temporally clustered packet drops.

Since the only indicator of congestion in RED is queue length, RED cannot really determine the severity of congestion because it is unaware of the number of competing connections (N). The lack of knowledge of N can cause congestion notification to be either too aggressive or not aggressive enough. Generally, when a bottleneck link is shared between N users, congestion notification to one connection reduces the offered load by a factor of $(1 - \frac{1}{2N})$ [2]. This can be a paltry drop of 0.05% when 100 TCP connections share the link, while resulting in a severe 25% drop when there are only two connections.

BLUE [1] uses packet loss and link idle events in addition to queue size to avoid congestion. BLUE varies the marking probability, P_m, depending on the severity of packet loss. However, P_m is decreased if the link is observed to be idle, thus maintaining high link utilization.

While maintaining reasonably low packet loss and high link utilization, BLUE causes the build-up of large queues which leads to increased queueing delay. Also, BLUE, like RED, does not take into account the number of connections multiplexed across the link in its congestion notification scheme.

In our AQM scheme, we take both the number of flows and the *RTT* of each flow into account to calculate the congestion-notification probabilities. We shall see how such a scheme nearly eliminates queueing delay, and yet maintains high link utilization and low packet loss. After presenting in our algorithm and experimental results in Sections III and IV respectively, we explain how the flow parameters in our algorithms can be estimated at the routers in Section VI.

[1]Henceforth, by *marking*, we mean explicitly setting a congestion bit in the packet header or dropping the packet.

[2]This is a frequently observed phenomena where sources increase and decrease their congestion windows at the same time.

III. Algorithm

Our AQM scheme applies the knowledge of the steady-state behavior of TCP connections at the router to intelligently mark packets for congestion notification. By using such a mechanism, a router provides each connection its fair share of throughput while preventing packet queues from building up.

In addition to available network bandwidth, a TCP connection's throughput depends on its round-trip time and the probability with which its packets are dropped in the network. Specifically, Mathis et al. [7] show that a connection's throughput at steady state satisfies the following equation under certain simplifying assumptions:

$$BW = \frac{MSS \times c}{RTT \times \sqrt{p}}$$

where BW is the throughput of the connection, MSS is the maximum segment size, RTT is its round trip time and p is the packet loss probability. The constant c depends on the acknowledgement strategy that is used (delayed or every packet) as well as on whether packets are assumed to be lost periodically or at random.

Now, let us consider a scenario where there are N active flows at a router on a particular outgoing link of capacity L. In our algorithm, a flow is considered to be active if it has at least one packet queued at the router for service or the packet for that flow was recently marked for congestion notification. Assuming that each flow is transmitting at a rate greater than or equal to $\frac{L}{N}$, then the fair share throughput of each flow is $\frac{L}{N}$. Substituting this for bandwidth in the above equation, we get an expression for loss probability p. Thus, p_i for flow i is:

$$p_i = \left(\frac{N \times MSS_i \times C}{L \times RTT_i}\right)^2$$

Note that this value of p_i may exceed 1, and so we modify the definition of p_i to preserve mathematical consistency to

$$p_i = min\left[1, \left(\frac{N \times MSS_i \times C}{L \times RTT_i}\right)^2\right]$$

By using this value of p_i as the marking probability for congestion notification, GREEN *coerces* each flow into sending at its fair rate. Note that unlike RED or BLUE, where a single marking probability (p_m) is used for every flow, the marking probability in GREEN is generally different for each flow because it depends on characteristics that are flow specific (MSS_i, RTT_i). Further, because p_i depends on the number of flows, we get the desired property of congestion notification being more aggressive for large N and less aggressive for small N.

Often, TCP connections with smaller RTTs are able to get better throughput than those with longer RTTs, since their window-increase cycles are clocked at a faster rate. By including RTT as an inverse parameter in our calculation, we are able to eliminate this bias of TCP against connections with longer RTTs.

IV. Experiments and Evaluation

A router is assumed to have knowledge of the bandwidth (L) of all outgoing links. N is the number of flows that have at least one packet queued at the router or whose packet was just dropped for congestion notification. The MSS of a flow is estimated by the router on a per-packet basis (assuming that most packets of a flow would be of the same size). By adopting such an approach, as opposed to averaging packet sizes from a particular flow over a certain interval, we reduce the need for flow state.

In our experiments we fix packet sizes at 1KB in all cases. The value of c, in our model (random dropping & delayed acknowledgement) is set to 0.93 [7]. For the time being, we assume that the router has some way of determining a flow's RTT. Later on, in Section VI, we shall see how this can be estimated.

To evaluate the performance of GREEN, we run a number of experiments using the Network Simulator *ns* [9] over the network topology shown in Figure 1. Sources and sinks connect to the routers over 10Mbps links with propagation delays ranging from 1ms to 50ms. The bottleneck link has a bandwidth of 1Mbps and a one-way propagation delay of

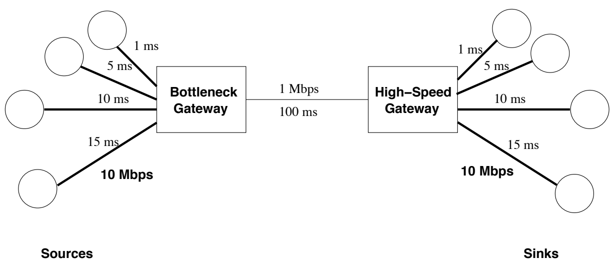

Fig. 1. Simulation Topology

100ms. Up to 500 FTP connections from the leftmost nodes (sources) to the rightmost nodes (sinks) start randomly within the first second of simulation and run for 100 seconds. The bottleneck gateway capacity is *one-tenth* the capacity of each of the individual links. By choosing such a configuration, we can evaluate the robustness of GREEN in an extremely congested environment at the bottleneck gateway. All of the performance metrics presented in the following sections – link utilization, packet loss and queueing delay – are measured at this bottleneck gateway.

For comparison purposes, we evaluate the performance of GREEN against RED and BLUE over the same topology. In the first set of experiments, we simulate all the algorithms using packet dropping rather than marking for congestion notification. In Section V we look at the results with packet marking using Explicit Congestion Notification (ECN) [3].

The queue weight in RED, W_q, is set according to the following equation [5]

$$W_q = 1 - e^{\frac{1}{C}}$$

where C is the link capacity in packets per second. Using a packet size of 1KB and a link capacity of 1Mbps, we set the value of W_q to be 0.008. Further in RED, queue sizes tend to stay around $Q_{minthresh}$ in lightly congested situations and around $Q_{maxthresh}$ in heavily congested ones [4]. Thus, to make congestion notification of RED (in our heavily congested simulation scenario) more aggressive, we set $Q_{minthresh}$ and $Q_{maxthresh}$ to 5 and 15, respectively. In BLUE, the freeze-time is set to 100ms, and the p_m increment and decrement values are set to 0.0025 and 0.00025 respectively. In other words we intentionally *pre-configure* RED and BLUE to handle our heavily congested simulation. In contrast, GREEN requires no such pre-configuration.

For GREEN, we introduce two parameters - $Q_{minthresh}$ and s_{level}. $Q_{minthresh}$, as in RED, is the minimum threshold of average queue size, below which packets are not marked for congestion notification. This additional parameter preserves link utilization, particularly when N is low. However, unlike RED, $Q_{minthresh}$ is an *auto-tuning* variable quantity and depends inversely on the number of flows. Specifically,

$$Q_{minthresh} = \frac{Q_{size}}{N}$$

Thus, a large number of active flows results in a lower threshold, allowing us to pursue congestion notification more aggressively. Large N prevents the queue from draining in spite of aggressive marking, and thus link utilization is maintained at a high level. Conversely, when there are a small number of flows, $Q_{minthresh}$ is higher and packet dropping

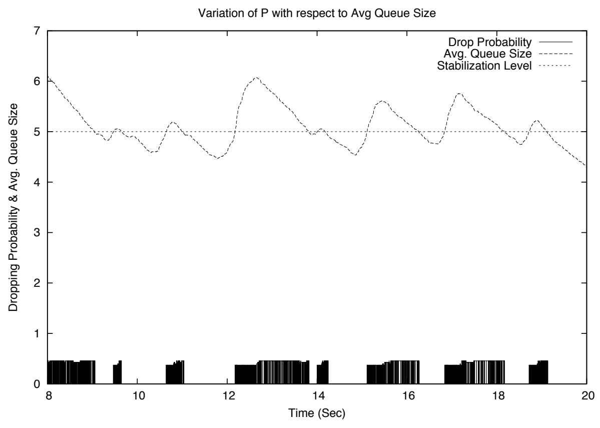

Fig. 2. An illustration of how s_{level} is equivalent to $Q_{minthresh}$

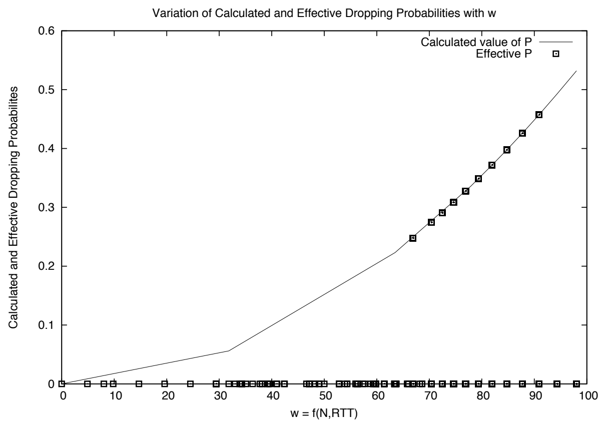

Fig. 3. Behaviour of Calculated and Effective Drop Probability w.r.t $w = f(N, RTT)$. Effective Drop Probabilty is zero as long as $Q_{avg} < s_{level}$.

is, therefore, pursued less aggressively. The second parameter, s_{level}, or *stabilization level* is the value at which queue sizes are to be stabilized. This value is set manually and independently from $Q_{minthresh}$. Note that this is not a performance-tuning parameter but a desired level of queue stabilization depending on how much average queueing delay can be tolerated by the end hosts. Queues are deterministically stabilized at s_{level}, irrespective of the number of flows. Packets are not marked for congestion if the average queue size remains below s_{level}. The difference between $Q_{minthresh}$ and s_{level} is that if the computed value of $Q_{minthresh}$ falls below s_{level}, then the *effective* $Q_{minthresh}$ is set to s_{level}. Thus, we have:

$$Q_{minthresh_{eff}} = max(Q_{minthresh}, s_{level})$$

In most cases (i.e., when N is large), $Q_{minthresh_{eff}}$ $= s_{level}$. Summarizing,

$$P_{i_{eff}} = \begin{cases} 0 & \text{if} \quad Q_{avg} \leq Q_{minthresh_{eff}}; \\ p_i & \text{if} \quad Q_{avg} > Q_{minthesh_{eff}}; \end{cases}$$

Figure 2 illustrates the equivalence property of s_{level} and $Q_{minthresh}$. The variation of the marking probability, p_i, (plotted on the x axis) is shown with respect to average queue size. We note that p_i is always 0 as long as Q_{avg} remains below s_{level}. However, the value of Q_{avg} does not affect p_i when Q_{avg} is greater than s_{level}. This value is calculated from the equation given in the previous section. Figure 3 shows the behaviour of the theoretical dropping probability (based solely on the GREEN equation) and the actual drop probability (shown as boxes in the graph) with respect to the variable parameters in the GREEN equation (N,RTT_i). The actual drop probability is either 0, when $Q_{avg} \leq s_{level}$, or lies along the theoretical p_i curve when $Q_{avg} > s_{level}$.

Figure 4 illustrates the behavior of different AQM schemes for 100 FTP sources, where s_{level} is set to 10 packets. For GREEN, this results in the queue size hovering around 10 packets, while

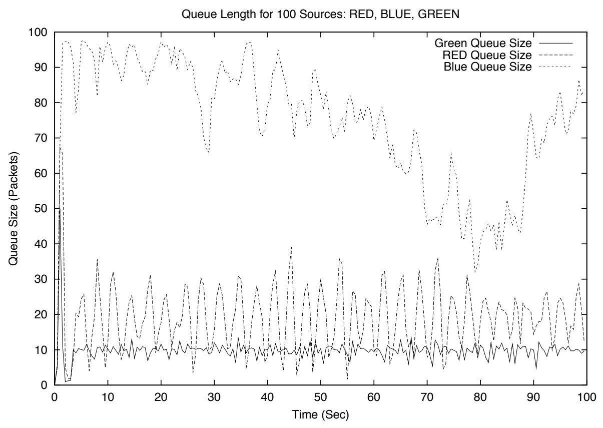

Fig. 4. Evolution of Queue Lengths over time for RED, BLUE and GREEN ($s_{level} = 10$) for 100 Sources.

much higher values and rapid fluctuations are observed in the case of RED and BLUE. Link utilization stays at above 98% for GREEN and above 99% for RED and BLUE. Figure 5 shows the corresponding packet-loss rates. Loss rates are comparable to RED but slightly higher than BLUE. The loss rates for GREEN, however, can be reduced by setting s_{level} to a higher value resulting in congestion notification that is less aggressive. Even when we increase the number of sources to 500, the queue size for GREEN remains at s_{level}. In Section V we explore how packet losses can be virtually eliminated using ECN.

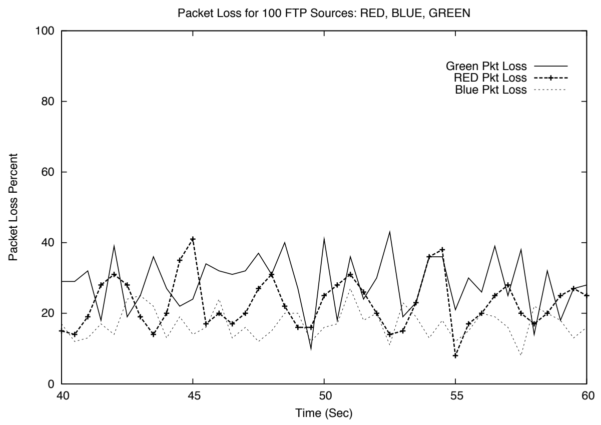

Fig. 5. Packet Loss, RED, BLUE and GREEN, 100 Sources.

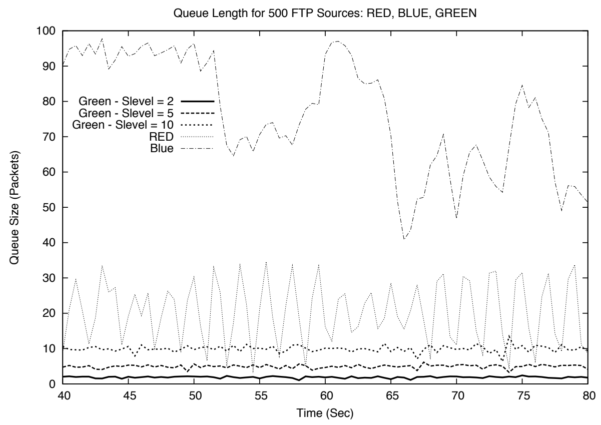

Fig. 6. Stabilized Queue Lengths for GREEN (with different s_{level} values) Compared to RED and BLUE for 500 Sources.

A. Discussion

Intuitively we can understand the performance improvement of GREEN as follows. The router, in calculating the marking probability of a flow, implicitly awards the flow its fair share of bandwidth. This allows the packet marking dynamics at the router to be "in synch" with the sending rate of the flow. Queues build up when the aggregate sending rate from all flows is more than the bottleneck bandwidth. By dropping packets based on TCP's congestion behavior, the router coerces the flows into sending at their fair rate, preventing queue build-up and stabilizing the queue at s_{level}. Remember that s_{level} in most cases (unless N is very low) is functionally equivalent to $Q_{minthresh}$ i.e., packets are not dropped if Q_{avg} is less than s_{level}. Hence, the queue is allowed to grow to s_{level} before congestion notification commences. Subsequent to this, flows send at their fair rate, stabilizing the queue at s_{level} after some initial fluctuation. The time to attain equilibrium depends on the feedback delay of the network path. Thus, the s_{level} parameter effectively adds an offset to the queue length below which it is not allowed to drain. In the next section, we present more results of this deterministic tuning property of GREEN.

B. Tuning Queue Sizes

For many types of application, especially delay-sensitive ones such as real-time audio and video, being able to control queueing delay is obviously a significant benefit. By varying the value of the stabilization level parameter, s_{level}, we can deterministically fix the queueing latency for flows that traverse the router. To illustrate this property, we set s_{level} to 2, 5 and 10 packets in three subsequent experiments. Figure 6 shows how the number of packets at any given time in the queue is precisely controlled by the value of s_{level}. Queue sizes are almost negligible in comparison with RED, where wide fluctuations are seen, and BLUE, where frequent transitions to drop-tail mode occur due to queue overflow. Link utilization in all three cases generally exceeds 90%, with link utilization close to 100%

for GREEN when $s_{level} = 10$. The latter result is not surprising since s_{level} actually controls the aggressiveness of the congestion-control behavior of GREEN. A higher threshold of 10 packets results in nearly 100% link utilization while a lower threshold of two packets drops utilization to around 90%. This behavior represents a trade-off between link utilization and queueing delay. We leave this as an implementation choice – a parameter that can be easily set by the network service provider.

We note that even for 100% link utilization, the queue size remains at a very low value (10 packets in this case) – an acceptable situation in most scenarios. Furthermore, the variation in queue size is very small which effectively eliminates delay jitter – an important and beneficial property to delay-sensitive traffic, which can be used in conjunction with QoS mechanisms where upper bounds on guaranteed delay can be provided to the end user.

However, the tunability of s_{level} is not infinite, i.e., the queue size cannot be stabilized to arbitrarily high levels. In particular, there exists a certain threshold τ above which the stabilization mechanism breaks down. τ turns out to be the bandwidth-delay product of the network path that the packets traverse from source to destination. Trying to fix the queue size above τ destroys the queue-size stability leading to oscillatory behavior. Stability is attained when aggregate sending rates are below the link capacity. Setting s_{level} greater than τ violates this condition in that the time taken to drain the queue from such a level is more than the feedback delay of the control loop. This results in an aggregate sending rate that is greater than the bottleneck bandwidth, causing queue build-up. For the simulation topology used in this paper, the value of τ is approximately 25 packets. Figure 7 illustrates how the stabilization property for queue behavior breaks down as we increase s_{level} to greater than τ.

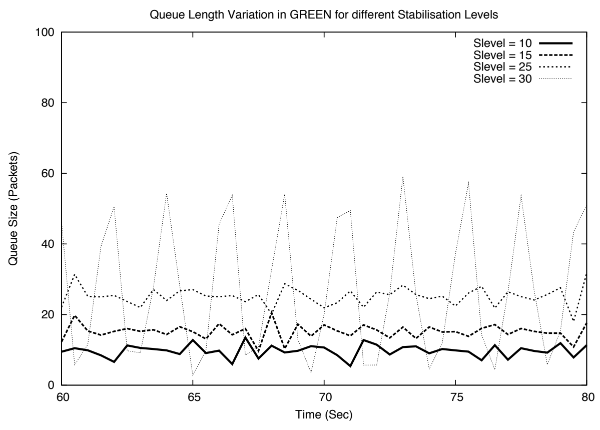

Fig. 7. An Illustration of Stabilization Break Down for $s_{level} \geq \tau$ ($\sim$ 25 packets) for 100 Sources.

V. GREEN with ECN

In Figure 5, packet loss rates for GREEN averaged 30% in the steady-state. Revisiting our equation, namely,

$$p = \left(\frac{N \times MSS \times C}{L \times RTT}\right)^2$$

we see that the dropping probability increases quadratically with the number of active flows. This leads to high packet loss rates in heavily congested scenarios such as ours. Although loss-rates for GREEN are comparable to RED 5, and queue-size stability is achieved, we seek to address this problem through *explicit congestion notification* (ECN) [3] instead of packet dropping as the means for congestion notification. In ECN, the IP header of a packet from an ECN-capable host contains a bit that is set by the router when it detects incipient congestion, or in our case, for proactive congestion notification. A host, upon reception of an ECN signal,[3] responds in the same way that it would to a lost packet, i.e., reduce its congestion window by half. Thus, ECN reduces packet losses and the associated retransmitted traffic. While using ECN leads to much lower loss rates, we can intuitively see how it would adversely affect queue stability. Inspecting Figure 5, we find that the packet loss rate for GREEN without ECN was about 30%, i.e., approximately a *third* of the packets were dropped for the purpose of congestion notification. In a scenario where ECN is used, all these dropped packets would be marked and then *queued*, resulting in greater queue sizes, and a greater number of active flows. This causes the marking probability p to increase (quadratically), causing aggressive marking, and hence, leading to fluctuating queue sizes. Figure 8 compares packet loss rates for RED, BLUE and GREEN, where ECN has been enabled for all sources as well as at the bottleneck gateway. By using ECN,

[3]The ECN signal from the router is conveyed to the sending host by an IP header bit that is set by the receiving host in the ACK packet.

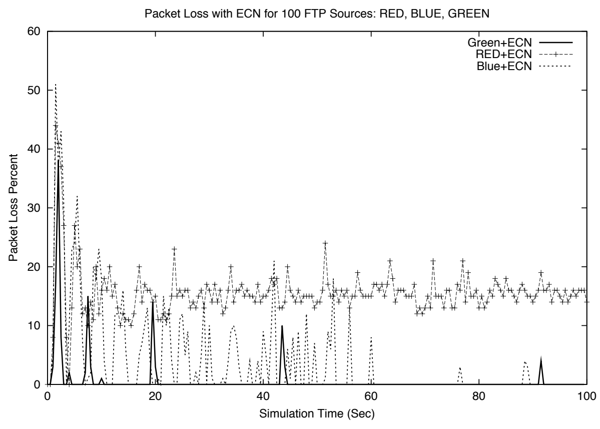

Fig. 8. Comparison of Packet-Loss Rates for ECN-enabled RED,BLUE and GREEN for 100 Sources.

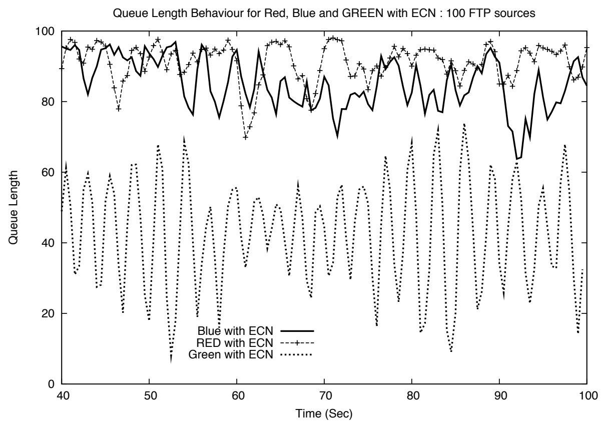

Fig. 9. Evolution of Queue Length for ECN-enabled RED, BLUE and GREEN for 100 Sources.

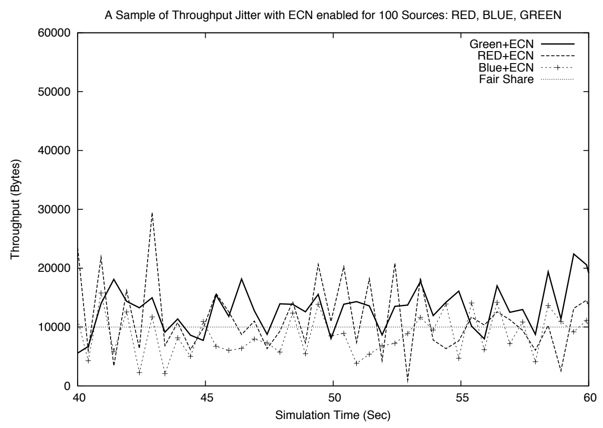

Fig. 10. Throughput Variation for ECN-enabled RED, BLUE and GREEN for 100 Sources.

GREEN eliminates almost all packet loss, whereas RED demonstrates a 15% loss rate and BLUE averages a 5% packet loss with peaks as high as 20%. As expected, however, queue sizes in ECN-enabled GREEN are prone to fluctuation (Figure 9). Queue sizes, and consequently, path delays are still well below those for either RED or BLUE. Applications are often able to maintain smooth output in the presence of delay jitter through buffering, as long as *throughput* remains more or less the same (Figure 10). We note from previous results that even in the absence of ECN, GREEN performs favorably compared to RED and BLUE, albeit with slightly higher packet loss. Since ECN is not widely deployed in today's Internet, we believe that this is another important advantage of GREEN over the other AQM schemes.

VI. Estimating Flow Parameters

Up until now, we assumed that the router had knowledge of the flow parameters required to calculate p, namely L, MSS, N, C and RTT. In this section, we briefly explain how the router can estimate these values.

L: Link Bandwidth. A router is assumed to have knowledge of the bandwidth capacity of all its attached links.

MSS: Maximum Segment Size. MSS is flow specific and can be estimated by maintaining flow state where the segment sizes are averaged over the duration of the flow. Flow state can be maintained in a method that is similar to *Stochastic Fairness Queueing* [8], whereby each flow is hashed onto a particular memory location. Hash collisions are ignored, thus keeping the time required for reading and updating flow state at $\mathit{0}(1)$. Alternatively, we can calculate this on a per-packet basis assuming that most incoming packets of a flow are likely to be of the same size. In our experiments we chose the latter method since packet sizes were fixed.

N: Number of Active Flows. The router determines N by hashing flows onto the flow-state structure. Each time a packet is queued or dequeued for that flow, a counter variable (which keeps track of the number of packets queued for that particular flow) is updated. When the counter reaches 0, the number of active flows is decremented. When a new flow starts and its packet is queued, it will be hashed onto an "inactive" location, and the number of active flows is subsequently incremented.

c: Modeling Constant. Depending on the assumptions in the model, c generally varies from 0.7 to 1.3. In our experiments (delayed mack and random loss), we set the value to 0.93 [7].

RTT: Round-Trip Time. This is, by far, the most difficult parameter to estimate with any degree of accuracy. Below we briefly present two solutions. The first solution involves a minor modification to the end-host software, while the second one involves router modifications. For a more detailed explanation of these mechanisms, the reader is referred to [12].

The first scheme makes use of the fact that the best estimates of RTT occur at the TCP end host. Using the IP options available, the sender can insert the RTT into a pre-specified field in the IP header. This value can then be utilized by routers along the way where GREEN is deployed. The advantage to this method is that it involves only a small change to the end-host software and can be easily deployed. The disadvantage of this method is that many non-TCP transport layer protocols like UDP, may not have any built-in mechanisms for estimating RTT. In such a case, routers may "punish" such non-conforming flows by awarding them the *smallest* observed RTT (since P_m is inversely proportional to RTT) among all flows. We believe that this punishment feature and the low-latency property of GREEN (mainly benefitting UDP-based applications such as real-time multimedia) would be incentives for UDP and similar protocols to keep a measure of their RTT values and insert these into the outgoing IP headers, as well as to adopt TCP-friendly congestion-control mechanisms. Although senders can also provide false information regarding their RTTs, and thus, get greater than fair share of bandwidth, this would require tampering with the protocol stack at the end-host software. While such a situation may not be a large-scale problem, it still remains a security issue.

The second scheme involves a small degree of work at the routers. The details of this scheme are beyond the scope of this paper, and the reader is referred to [12]. The basic idea is to have each router add the propagation delay of the outgoing link to a pre-specified field in the IP header. This field is initialized at the edge routers and is incremented at each hop. Assuming symmetric paths, routers will have partial one way trip-times (OTTs) of the data packets going from source to destination and of ACK packets in the reverse direction. These values are hashed onto the same memory location using commutative hash functions on (source,dest) IP-address pair. Adding the two partial OTT values and doubling the result will give us the RTT for that flow with some error caused due to the difference in queueing delays between the data packets and acknowledgement packets. However we would still have a much better estimate of RTT than by simply doubling the delay of the attached outgoing link. This method however fails to work in the presence of path asymmetries, which have been shown to exist in the Internet with increasing frequency [11]. In such a case, the best the router can do is to double the partial OTTs for an RTT approximation.

VII. Conclusion

In this paper, we have shown how applying the knowledge of steady-state TCP congestion control behavior at the routers helps prevent congestion in the network much more effectively. We found that we are able to deterministically stabilize the queue size, while maintaining optimal throughput and link utilization. Using this algorithm, GREEN can fix an upper bound on queueing delay – a useful property for implementing QoS schemes. Packet-loss rates can further be reduced when ECN rather than packet-dropping is used for congestion notification. Future investigations will include how the mechanism behaves for short-lived connections (so-called "web mice") with mixtures of different traffic types and in the presence of two-way traffic. The applications that stand to benefit the most from the low-latency/high-throughput property of GREEN are delay-sensitive traffic such as streaming real-time audio and video. Since many of these are UDP-based, we believe that this benefit provides incentive for UDP and similar transport layer protocols to adopt TCP friendliness in their congestion-control mechanisms.

References

[1] W. Feng, D. Kandlur, D. Saha, K. Shin. BLUE: A New Class of Active Queue Management Algorithms. In *UM CSE-TR-387-99*, April 1999.

[2] W. Feng, D. Kandlur, D. Saha, K. Shin. A Self-Configuring RED Gateway. In *Proceedings of IEEE INFOCOM*, March 1999.

[3] S. Floyd. TCP and Explicit Congestion Notification. *Computer Communication Review*, 24(5):10-23, October 1994.

[4] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[5] S. Floyd, R. Gummadi and S. Shenker. Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management. *Under Submission*, August 2001.

[6] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM 1988*, pages 314–332, August 1988.

[7] M. Mathis, J. Semke, J. Mahdavi, T. Ott. The Macroscopic Behavior of the TCP Congestion-Avoidance Algorithm. *Computer Communication Review*, Vol.27, No.1, July 1997.

[8] P. McKenney. Stochastic Fairness Queueing. In *Proceedings of IEEE INFOCOM*, March 1990.

[9] Network Simulator. *http://www.isi.edu/nsnam/ns.*

[10] T. Ott, J. Kemperman, M. Mathis. The Stationary Behavior of Ideal TCP Congestion Avoidance. *ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps*, August 1996.

[11] V. Paxson. End-to-End Routing Behaviur in the Internet. *IEEE/ACM Transactions on Networking*, Vol.5, No.5, October 1997.

[12] S. Thulasidasan. An Algorithm for Estimating End-to-End Path Delays at Routers. *In Progress.*